The FaultCheck distribution

This collection of software consists of the tool FaultCheck, which provides fault injection support for property-based testing tools, and one use case. In the use case, a generic simple end-to-end library written in C is tested with ScalaCheck. There are also several property-based tests to evaluate whether FaultCheck fulfils its specification based on ScalaCheck.

Scripts

This distribution contains several scripts to automate certain tasks. Their purpose is as follows:

Directory: /

set_env
Set up the environment for running FaultCheck and the E2E example where the dynamic libraries are looked up and the includes are resolved.

build
Compiles FaultCheck and the C files for the example(s).

build_doc
Generates the documentation from the README.md markup file. The output appears in the *Documentation* directory.

clean
Removes the object files from the build process.

clean_scala
Removes all the bytecode files generated by the ScalaCheck example and the ScalaCheck tests.

Directory: /Examples/E2E_Scala

gen_coverage
Generate C code coverage reports in HTML.

clean_coverage
Remove the coverage files.

Installing dependencies

In order to run this software, make sure that all libraries that are required are installed. For FaultCheck itself, the Qt SDK is required and for the examples the GNU C compiler is required. For the scala example and the FaultCheck tests, the sbt tool is required. To generate the coverage reports and the documentation, some additional tools are required.

Ubuntu Linux

Setting everything up on a fresh Ubuntu install is fairly simple. First, install some dependencies:

```
sudo apt-get install build-essential qt-sdk qt4-qmake
```

ScalaCheck examples and tests
For the ScalaCheck examples and tests, install scala, openjdk and sbt:

```
wget http://apt.typesafe.com/repo-deb-build-0002.deb
sudo dpkg -i repo-deb-build-0002.deb
sudo apt-get update
sudo apt-get install sbt scala openjdk-7-jdk
```

Coverage information
To generate coverage information with the provided script for the examples, install lcov 1.10 or later. On Ubuntu 14.04, the version from the repositories can be used:

```
sudo apt-get install lcov
```

For earlier versions of Ubuntu, download and install an updated version of lcov from here:
http://packages.ubuntu.com/trusty/all/lcov/download

Documentation
To generate LaTeX, PDF and HTML documentation from the markdown file, install pandoc and some LaTeX tools:

```
sudo apt-get install pandoc texlive-latex-base cm-super
```

Setting up the environment

From the root directory, set up the environment with the provided scrip. The script has to be run with source so that it is executed from the current shell:

```
source set_env
```

Build everything:

```
./build
```

At this point, the ScalaCheck example and the functionality tests should work from the current shell. Note that the environment has to be set with the source command every time a new shell is started. Also note that the CopterSim scripts set up the environment themselves, so they can be run independently.

E2E example with ScalaCheck

There is one example where fault injection on a generic end-to-end library written in C is performed. The test is written in Scala and uses ScalaCheck.

Go to *Examples/E2E_Scala/scala* and run the test

```
sbt run
```

The first time this is done, the compilation and dependency fetching of sbt might take some time. After the compilation a lot of output is printed from the test. In the end, the test should pass.

To switch the E2E-library on and off, open *Examples/E2E_Scala/scala/src/main/scala/e2e/AirbagTest.scala* and change the variable near the beginning:

```
val useE2E
```

Switching the E2E-library off and running the test again should reveal some failing test cases.

Generate code coverage information

The *gen_coverage* script in the corresponding example can be used to generate coverage information in the *Coverage* directory. This can only be done after the experiment has been run. In *Examples/E2E_Scala/scala/src/main/scala/e2e/AirbagTest.scala* different faults can be switched on and off (see the comments) to exercise different or all fault handling mechanisms.

Testing FaultCheck

There are some property-based tests that can be used to verify that FaultCheck works as intended. These tests are implemented in ScalaCheck. When new features are added to FaultCheck, these tests should be updated and/or re-run. Currently, there are two different tests: one for the probing-part of FaultCheck and one for the packet-based communication channel of FaultCheck. For these tests to work, everything has to be built and the environment has to be set up as described above.

Testing the probing part

Go to *Tests/General* and run the test with

```
sbt run
```

All the tests should pass.

Testing the communication channel

Go to *Tests/Packet* and run the test with

```
sbt run
```

All the tests should pass.

Generating documentation

Currently, this tutorial is written with the markdown syntax in the *README.md* file and the files in the Documentation directory are generated with the *build_doc* script. To update the documentation, edit the *README.md* file and run the script:

```
./build_doc
```

Make sure that the dependencies for this script are installed as described above.